

# *Proxy Applications: Vehicles for Co-design and Collaboration*

PSAAP II Kick-off meeting

Albuquerque, Dec 10, 2013

Rob Neely



LLNL-PRES-647480

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC



# The role of proxy applications in exascale co-design

- **Co-design:** A collaboration between vendors, hardware architects, system software developers, domain scientists, computer scientists, and applied mathematicians **working together** to make informed evaluations about **hardware and software** components necessary for a successful transition to exascale
- **Proxy applications:** The “language of co-design”. They are simplified representations of algorithms, data motion patterns, and coding styles used to do early **evaluation of trade-offs** in the hardware and software design space

# Co-design Provides a Formal Methodology for us to Work With the Community

## What is Co-design?

- Deep collaboration... (See previous slide)

## How Does Co-design Work?

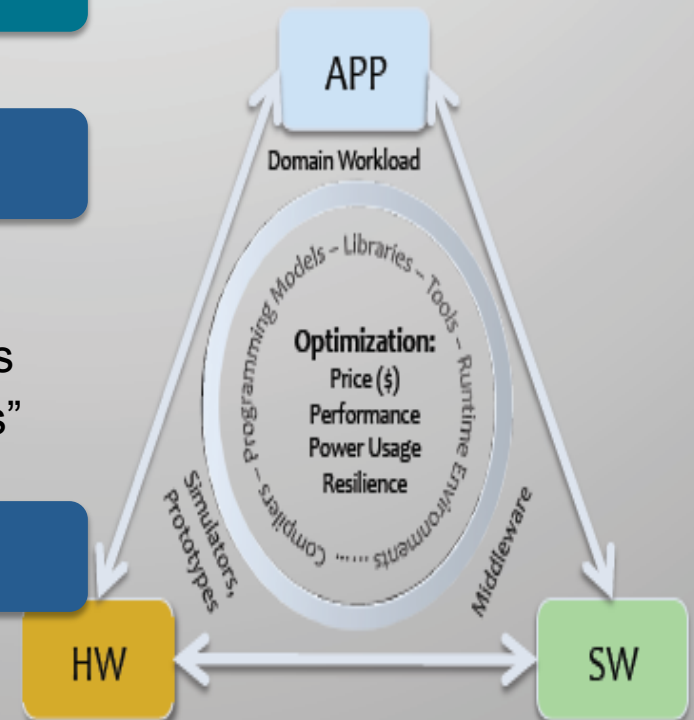
- Hardware vendors provide access to future roadmaps
- Researchers develop new software methodologies
- Application developers provide “proxy applications” for open study of software requirements

## How is this different from past practice?

- Time-frame (5-10 years)
- Co-dependence on successful HPC strategy

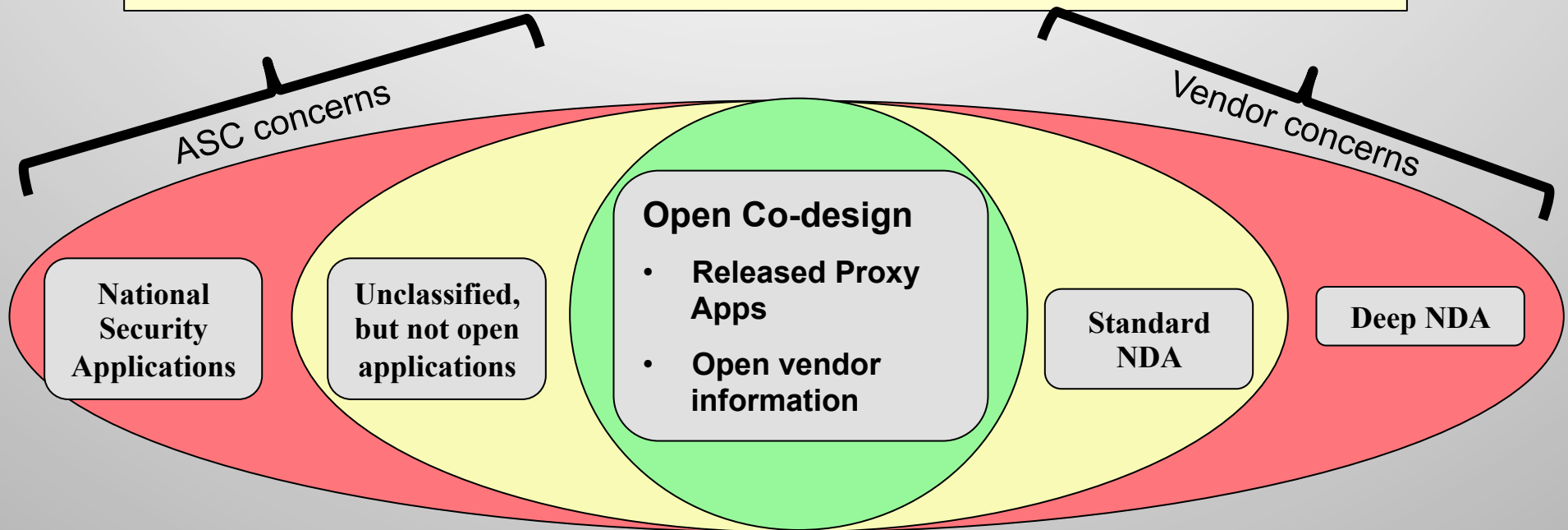
## What are the challenges of co-design?

- Unclassified proxy applications are required
- Deep NDA and trust with multiple vendors



# (One of) the difficulties of co-design

Co-design gets more difficult the further you get from open collaboration and the closer you get to the “truth” (particularly with competing vendors and national security applications)



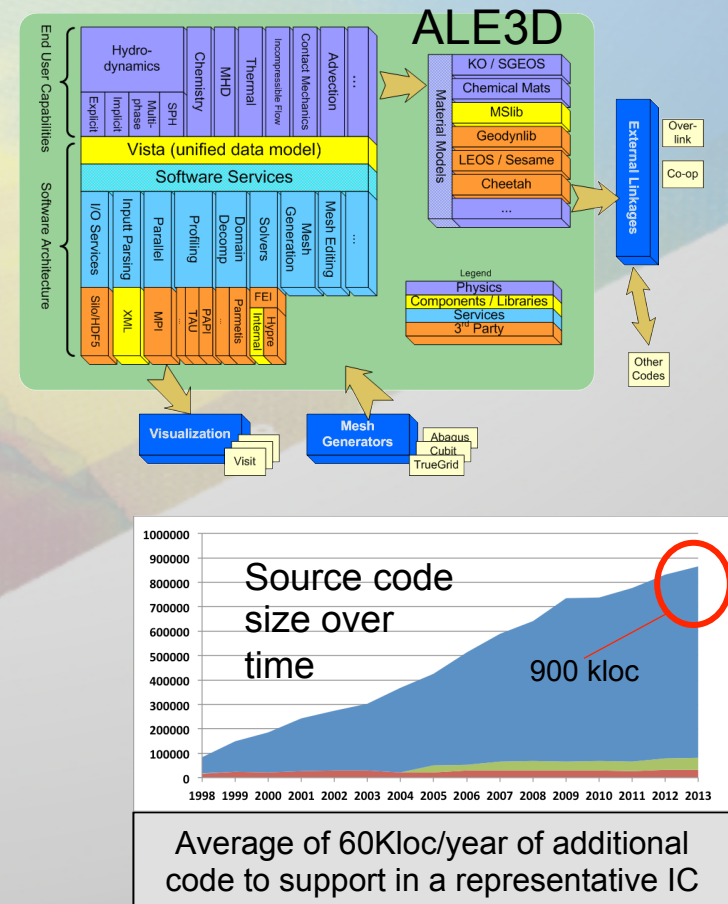
- **ASC** : Involve staff with clearances in co-design efforts and developing proxy apps
- **Vendor** : Limit number of lab staff engaging in multiple “deep NDA” discussions

# ASC NSApp Co-design Project

- ExMatEx, CESAR, and ExaCT are ASCR-funded co-design centers
  - 2 of 3 led at NNSA labs
  - Lots of opportunity for cross-fertilization between NNSA and Office of Science
- Co-design centers are designed to be application-centric
  - NNSA already has the applications, but they're typically not open (export controlled / classified)
- NNSA/ASC co-design effort - NSApp CDP
  - National Security Applications (LANL / LLNL / SNL)
  - See paper at [codesign.llnl.gov](http://codesign.llnl.gov)

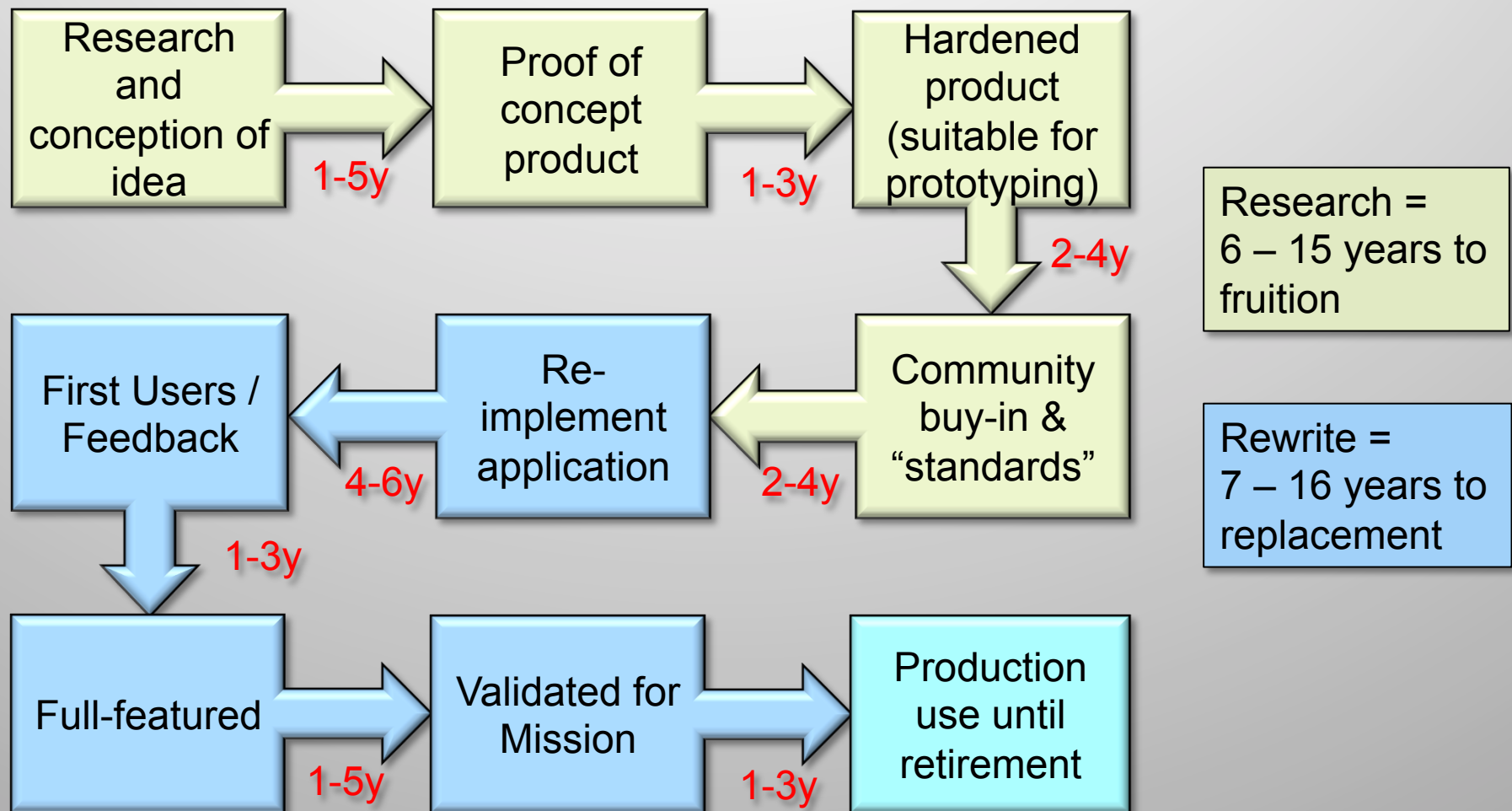
# ASC multi-physics applications are complex, expensive to maintain and evolve, and daunting to rewrite. Proxy apps to the rescue!

- Support broad ranges of applications
- > 10 packages, 10-30+ third party libraries
- Many different spatial, temporal scales
- Multi-language (C++, C, Fortran90, Python)
- Variety of parallelism approaches
- Long life-time projects with >1 million lines of code
- 15+ years of development by large teams (10 – 20+ FTEs)
- Steerable / interactive interfaces
- Algorithms tuned for minimal turn-around time (vs. max compute efficiency)
- How future physics model improvements will impact compute balance between packages is unknown

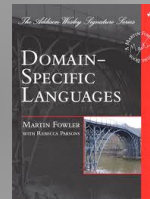
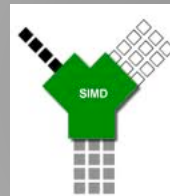


The difficulty is compounded by continuing to deliver the programmatic mission while addressing the challenges of next generation advanced architectures.

# Full Re-implementation of Large Codes Is a Decadal Process



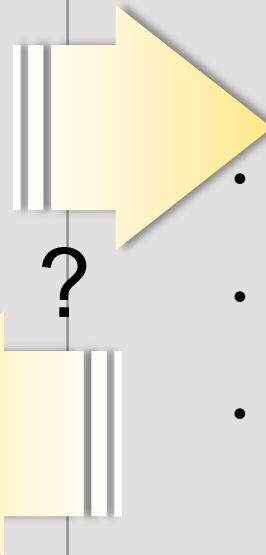
# Evolve or Rewrite? This is a fundamental question we're addressing



## Evolve existing code bases

- Gain experience with massive scaling (Sequoia / BlueGeneQ)
- Expose fine-grained concurrency
- Accelerator directives
- Application controlled resilience and power management
- Leverage validated code base

At a minimum, we know we're going to have to do this. But will it take us far enough?

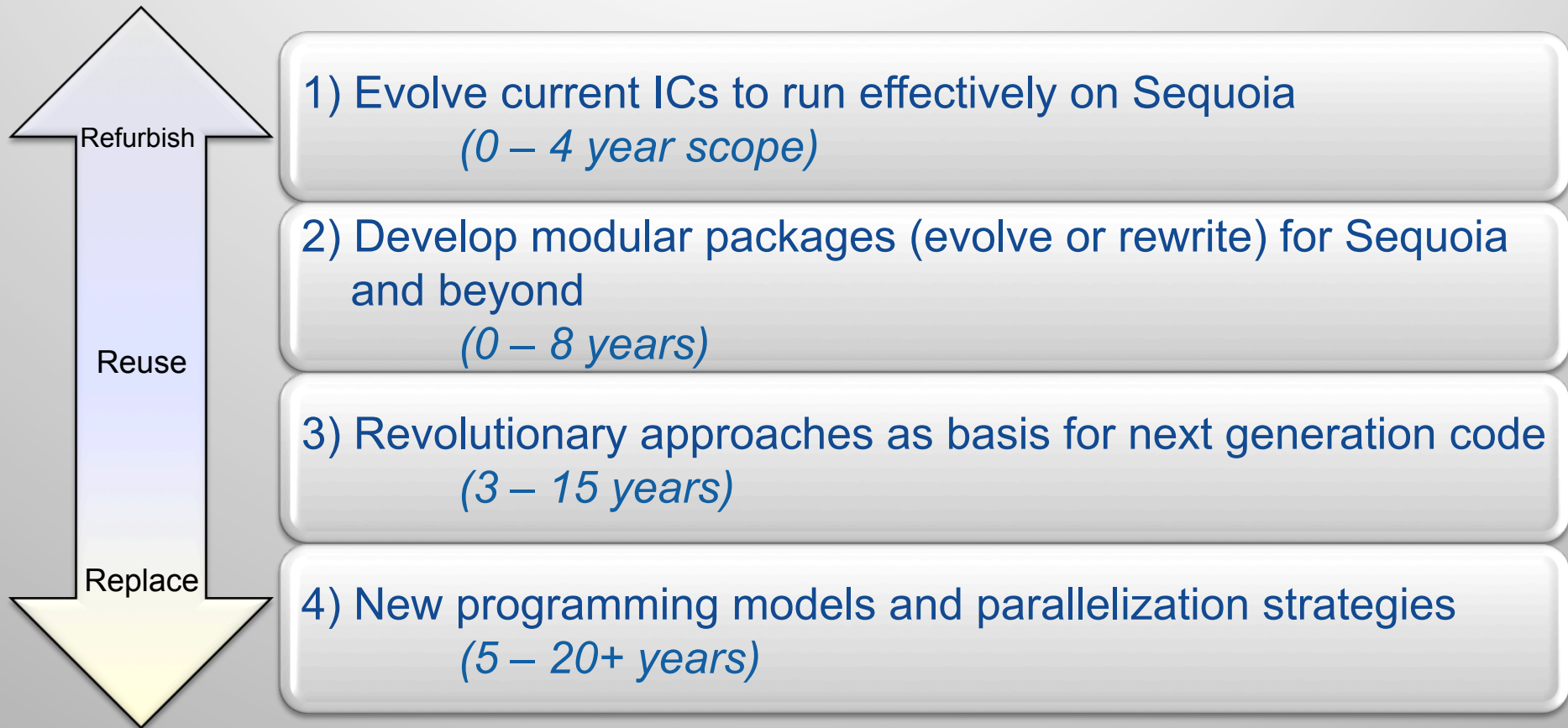


## Undertake new “from scratch” rewrite

- Evaluate and gain experience with new programming models
- “Harden” them beyond research prototype phase
- Determine degree of rewrite needed (if any)

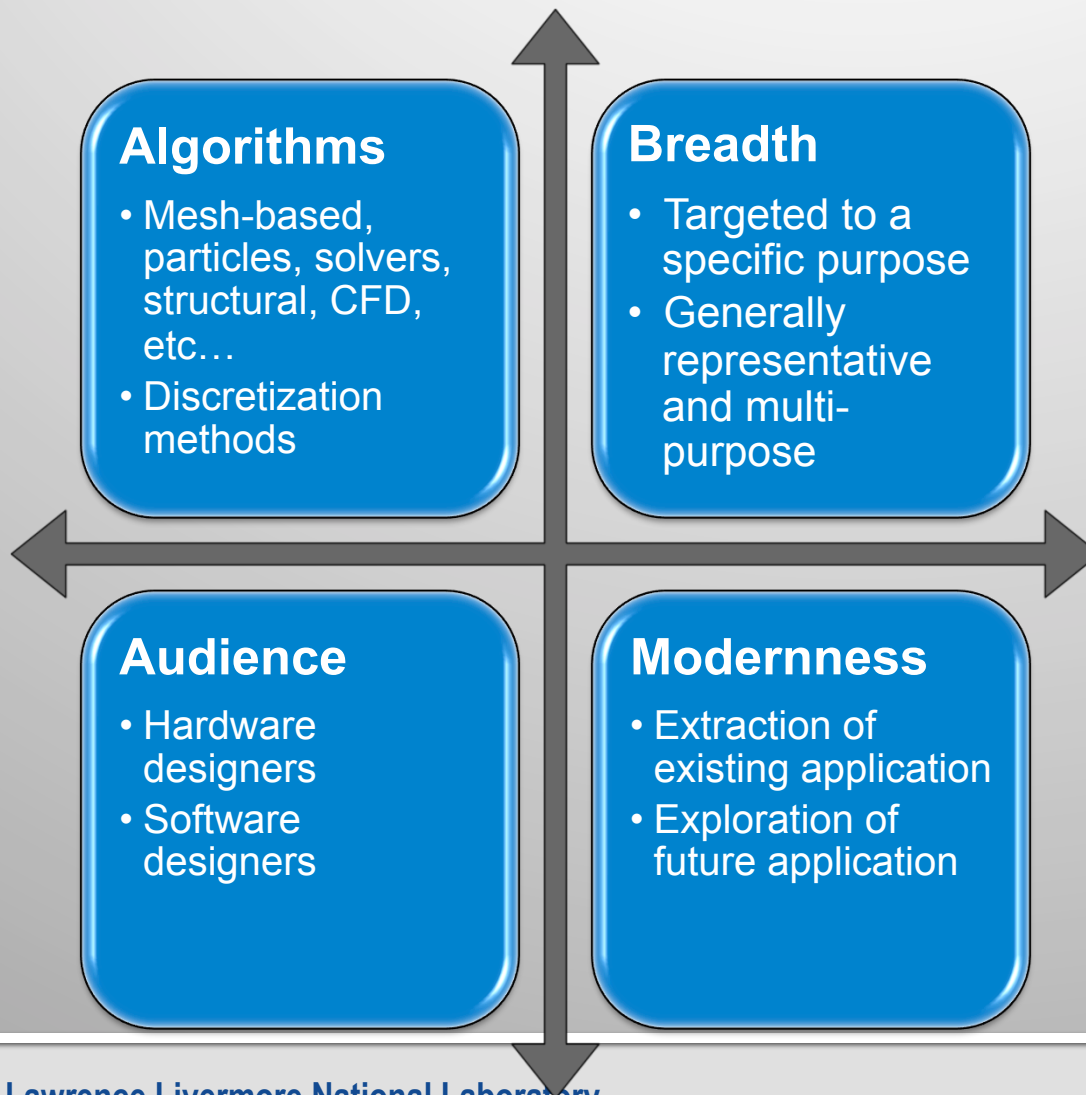
New languages and a clean slate approach are compelling, but can we manage the risk?

# LLNL ASC Integrated Code Strategy – Four “pillars” capture our exploration space



Our Challenges are not tied to the pursuit of exascale computing.  
Future architectures at extreme scale (100's of Pf) are just as demanding.

# A suite of proxy applications should span a wide design space



- One must consider a broad range of proxy applications to even begin to cover the design space of an ASC application
- Proxy apps are more than just a benchmark. They are meant to be modified – perhaps dramatically
- Interoperability of software solutions is key

# The \$65,536 dollar question

- What is the right balance between too big to understand, and too simple to be representative?



Small [ $O(1k \text{ loc})$ ]

**Pros** – Easy to pick up and learn

**Cons** – Hard to draw general conclusions from



Larger [ $O(25-75k \text{ loc})$ ]

**Pros** – Likely more representative of real applications

**Cons** – Some benefits are lost to added complexity and size

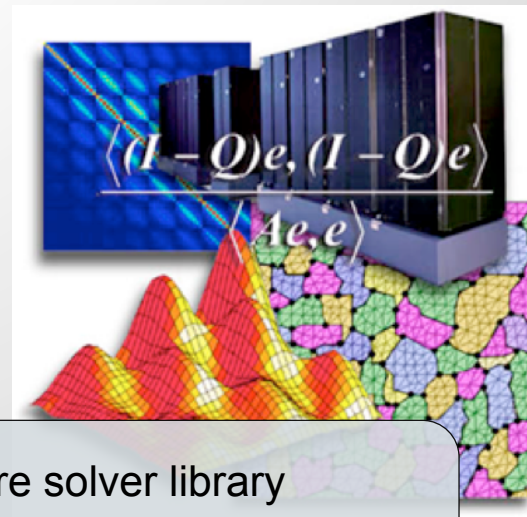
Understand: Small proxy applications are often gross approximations of reality. Be careful of the conclusions you draw!

# Sample LLNL success stories with proxy apps

- LULESH
  - Ported to 8+ programming languages – allowed an unprecedented comparison. (IPDPS Best Paper 2012)
  - Helped influence development of Chapel, Liszt, Charm++, ...
  - Used in ExMatEx co-design center to great effect in FastForward interactions
- LCALS
  - Allowed rapid identification of optimization issues with multiple vendor compiler teams.
- AMG / UMT / MCB / LULESH / SNAP
  - Repurposed as TN8/CORAL benchmarks (NNSA 2015-17 large scale procurements)
  - Co-design has influenced changes in how we do procurement benchmarking
- Lassen
  - First (?) example of Charm++ used as a library component

Numerous examples of proxy apps giving students and new hires a jump start

# AMG2013: Algebraic Multi-grid



## Description

- Derived from BoomerAMG in LLNL's hypre solver library
- Representative of implicit solves performed in large unstructured applications

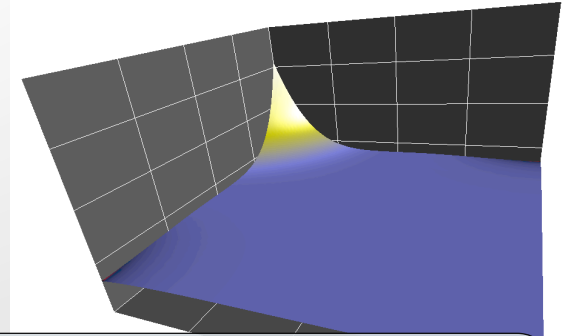
## Characteristics / Uses

- Stresses memory bandwidth
- Irregular communication patterns and memory accesses
- Fine-grained threading
- Acceleration

## Language(s) / Size

- C (~70k loc)
- MPI
- OpenMP

# MCB: Monte Carlo



## Description

- Little or no real physics – just a “particle pusher”
- Object-oriented design

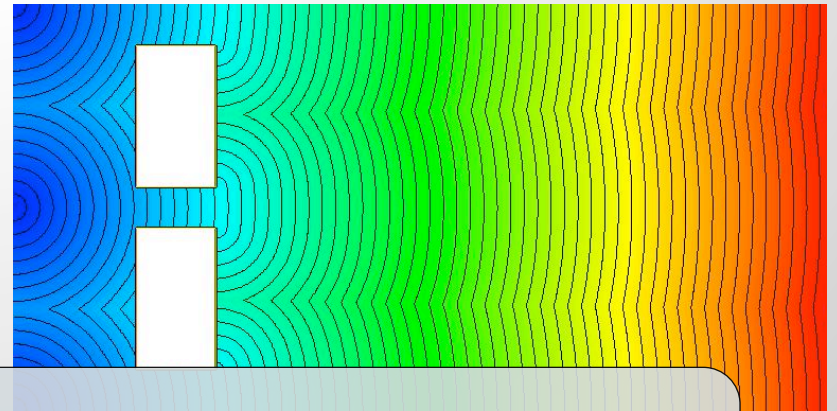
## Characteristics / Uses

- Low floating point intensity, large amount of branching
- Irregular memory and communication patterns
- Could be used for exploration of alternatives to MPI (e.g. PGAS), transactional memory,

## Language(s) / Size

- C++ (~13k loc)
- MPI
- OpenMP

# Lassen: Front tracking



## Description

- Front-tracking algorithm used to propagate wave-fronts and pre-calculate arrival times
- Work is isolated to a narrow region around the front

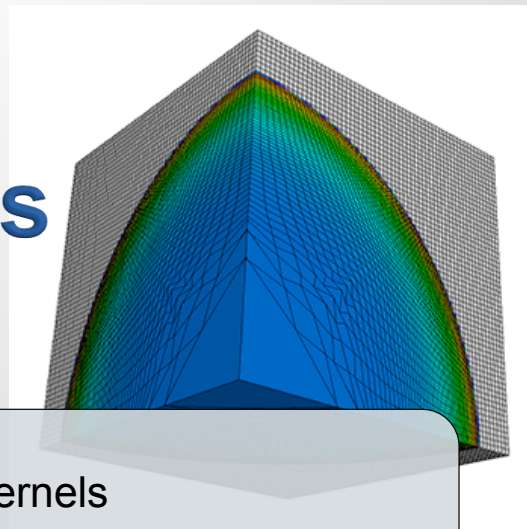
## Characteristics / Uses

- Highly load-imbalanced (1D front in a 2D mesh)
- Similar issues to sweep-based algorithms
- Studying task-based parallelism or dynamic load balancing
- Interoperability of MPI and other prog. models (e.g. Charm++)

## Language(s) / Size

- C++ (3,500 loc)
- Charm++
- MPI

# LULESH: Lagrangian Hydrodynamics



## Description

- Unstructured mesh
- Small in size – about 12 representative kernels
- Ported to numerous different programming models
- Version 2.0 released in 2013

## Characteristics / Uses

- Unstructured mesh data structures (indirection)
- Analysis of thread overheads in OpenMP
- Port to various programming models

## Language(s) / Size

- C++ (5k loc)
- MPI / OpenMP
- A++, Chapel, CUDA, OpenACC, Loci, Liszt, Charm++ versions available

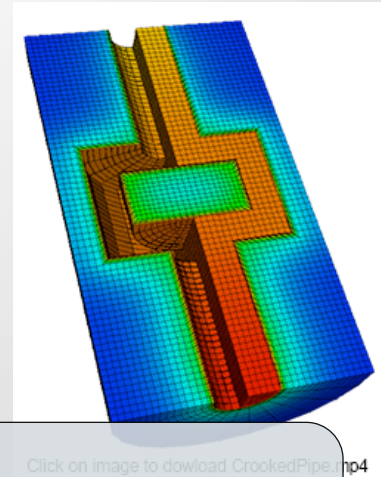
[illegible]

- Livermore Compiler Analysis Loop Suite
- Based on classic “Livermore Loops” from the 70’s-80’s
- ~30 representative loop kernels
- Easy framework to extract individual loops, add new ones

- Primarily designed to work with compiler vendors on optimizations (e.g. SIMD)
- Multiple versions: “Raw”, “Lambda/RAJA”, cilk plus, ...
- Some loops large enough to study threading / runtimes

- C++ (4k loc)
- OpenMP

# Mulard: Multigroup Rad Diffusion



## Description

- Includes some simpler mini-apps (Duckling, Hatchling) as well
- Built on top of MFEM library

## Characteristics / Uses

- Coupled implicit solve methods
- Use of abstractions in design of finite-element based apps
- Ability to modify spatial ordering of the mesh
- Useful to study acceleration, transactional memory, ...

## Language(s) / Size

- C++ (46k loc)
- OpenMP

# LIP (not yet released)

## Description

- 1D/2D tabular interpolation library
- Builds sets of tabular data for repeated lookups
- Basis for Equation-of-State calculations

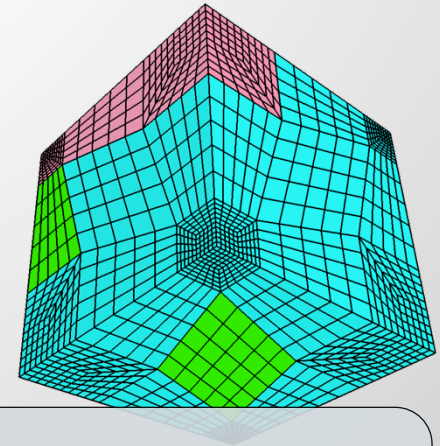
## Characteristics / Uses

- Designed to study memory hierarchies
- Sharing of data between MPI processes on same node
- Effective caching strategies for NUMA architectures
- Threading strategies

## Language(s) / Size

- C (11k loc)
- MPI driver to spawn multiple copies (no communication)

# LUAU3D (not yet released)



## Description

- 3D Unstructure mesh advection
- Complex data motion – fluxing of material through element faces
- No physics (velocities), just mesh relaxation and material flux





## Characteristics / Uses

- Useful counterpart to LULESH for understanding ALE (arbitrary lagrange eulerian)
- Irregular memory accesses, lots of p2p communication
- Element ordering, acceleration, ...

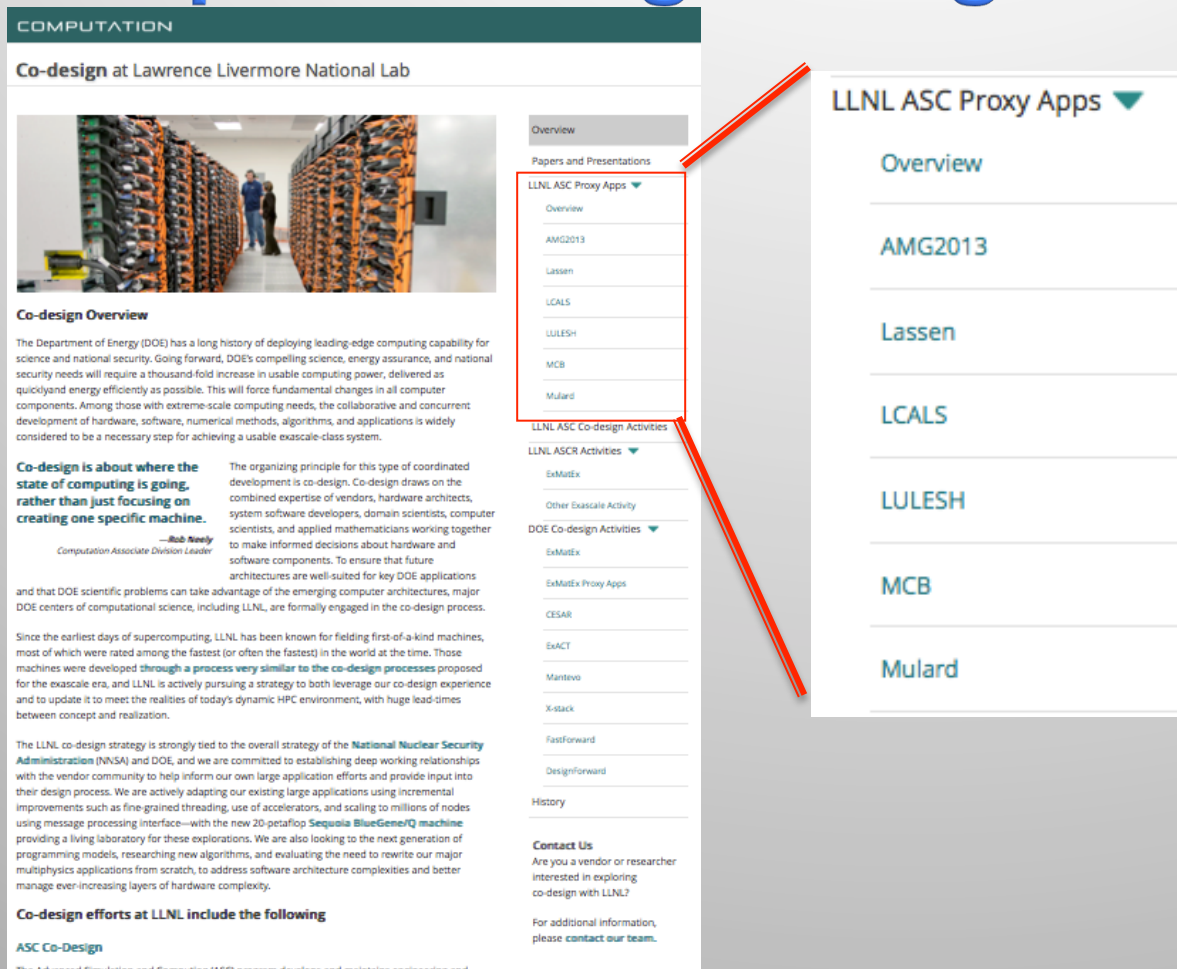
## Language(s) / Size

- C++ (26k loc)
- MPI
- OpenMP (in later versions?)

# LLNL ASC Proxy App Suite

Proxy	Type	Lang	LOC	Description	Example uses
<b>AMG2013</b>	Mini	C++, MPI, OMP	75k	Algebraic multi-grid. Irregular memory and comm. Subset of hydre solver library.	Investigating acceleration methods, thread performance, irregular network access patterns
<b>Lassen</b> 	Mini	C++, Charm++	3.5k	Front tracking through a 2D mesh. Work is concentrated at front, leading to high load imbalance.	Exploring task-based programming models and load balance strategies
<b>LCALS</b> 	Skel	C++, OMP	4k	Updated "Livermore Loops". ~30 loops in a common analysis framework.	Compiler optimizations. Vectorization and thread models.
<b>LULESH (v2.0)</b>	Mini	C++, MPI, OMP	5k	Lagrangian hydro on arbitrary connected hex mesh. 2.0 includes regions, and unified ser/par source.	New programming models and alternate languages. Overall performance characteristics of hydro
<b>MCB</b>	Mini	C++, MPI	13k	Monte Carlo Particle transport, Low floating point intensity. Lots of integer and branching	Threading, transactional memory, acceleration, irregular messaging, PGAS
<b>Mulard</b>	Mini	C++	46k	Implicit multigroup radiation diffusion. Built on MFEM library.	Matrix solution techniques. Finite element abstractions.
<b>LIP</b> 	Skel	C		Library for doing 2D interpolation on large tabular data	NUMA techniques. Sharing data between MPI tasks.
<b>LUAU</b> 	Mini	C++, MPI		Multi-material advection on arbitrary connected hex mesh	Impact of memory indirection, bandwidth intensive.
<b>UMT</b>	Mini	C++, C, F90, Py		Radiation transport. Unstructured mesh sweep.	Interconnects (large messages). Memory bandwidth and capacity.

# For more info: <http://codesign.llnl.gov>



**COMPUTATION**

## Co-design at Lawrence Livermore National Lab

**Co-design Overview**

The Department of Energy (DOE) has a long history of deploying leading-edge computing capability for science and national security. Going forward, DOE's compelling science, energy assurance, and national security needs will require a thousand-fold increase in usable computing power, delivered as quickly and energy efficiently as possible. This will force fundamental changes in all computer components. Among those with extreme-scale computing needs, the collaborative and concurrent development of hardware, software, numerical methods, algorithms, and applications is widely considered to be a necessary step for achieving a usable exascale-class system.

**Co-design is about where the state of computing is going, rather than just focusing on creating one specific machine.**

—Bob Newby  
Computation Associate Division Leader

The organizing principle for this type of coordinated development is co-design. Co-design draws on the combined expertise of vendors, hardware architects, system software developers, domain scientists, computer scientists, and applied mathematicians working together to make informed decisions about hardware and software components. To ensure that future architectures are well-suited for key DOE applications and that DOE scientific problems can take advantage of the emerging computer architectures, major DOE centers of computational science, including LLNL, are formally engaged in the co-design process.

Since the earliest days of supercomputing, LLNL has been known for fielding first-of-a-kind machines, most of which were rated among the fastest (or often the fastest) in the world at the time. Those machines were developed through a process very similar to the co-design processes proposed for the exascale era, and LLNL is actively pursuing a strategy to both leverage our co-design experience and to update it to meet the realities of today's dynamic HPC environment, with huge lead-times between concept and realization.

The LLNL co-design strategy is strongly tied to the overall strategy of the **National Nuclear Security Administration (NNSA)** and DOE, and we are committed to establishing deep working relationships with the vendor community to help inform our own large application efforts and provide input into their design process. We are actively adapting our existing large applications using incremental improvements such as fine-grained threading, use of accelerators, and scaling to millions of nodes using message processing interface—with the new 20-petaflop **Sequoia BlueGene/Q machine** providing a living laboratory for these explorations. We are also looking to the next generation of programming models, researching new algorithms, and evaluating the need to rewrite our major multiphysics applications from scratch, to address software architecture complexities and better manage ever-increasing layers of hardware complexity.

**Co-design efforts at LLNL include the following**

**ASC Co-Design**

The Advanced Simulation and Computing (ASC) program develops and maintains engineering and

**LLNL ASC Proxy Apps**

- Overview
- AMG2013
- Lassen
- LCALS
- LULESH
- MCB
- Mulard

**LLNL ASC Co-design Activities**

- LLNL ASCR Activities
  - ExMatEx
  - Other Exascale Activity
- DOE Co-design Activities
  - ExMatEx
  - ExMatEx Proxy Apps
  - CESAR
  - ExACT
  - Manitvo
  - X-stack
  - FastForward
  - DesignForward
- History

**Contact Us**

Are you a vendor or researcher interested in exploring co-design with LLNL?

For additional information, please [contact our team](#).

- Get latest versions of LLNL proxy apps
- Contact developers
- References to other co-design efforts

# How might this all relate to the PSAAP2 Centers?

- PSAAP centers have a unique opportunity to inject changes in NNSA code development processes
  - Exascale is driving disruptive changes.
  - Both sides (NNSA/Univ) must push exascale research concepts into complex applications
- Communicating CS concepts developed at your centers through simpler proxy apps is an effective method
- We invite you use our proxy apps
  - Developing a DSL or other abstraction? Test it on multiple proxy apps
  - Ask us questions – don't assume anything!
  - Communicate back your changes to us (we learn from them as well)
- We expect to engage in co-design with you
  - We can be a conduit to the vendor community for you

